# Layout-independent Fisheye Views of Nested Graphs

Emanuel G. Noik (noik@db.toronto.edu)

Computer Systems Research Institute
University of Toronto
Toronto, Canada M5S 1A1

## Abstract

*Although a graph can be a useful device for visualizing complex relationships, drawings of large graphs can be difficult to comprehend. As one remedy, we formulated a novel generalized approach for generating fisheye views of nested graphs with multiple variable focal points, and devised an algorithm that creates fisheye views in the absence of application specific distance metrics. Previous solutions produced fisheye views by filtering or distorting drawings of graphs. Since these approaches relied on geometric notions of distance, they could only be applied effectively in limited cases. By contrast, our approach treats fisheye view generation as a phase that precedes graph layout, rather than as a technique that alters an existing drawing, and does not suffer these drawbacks.*

## 1 Introduction

The graph topovisual formalism [6] is well-suited to the task of visualizing complex relations on a set of elements [2, 13, 15, 17, 20]. Large information spaces, however, must be depicted by large graphs, thereby diminishing the effectiveness of the graph as a visualization device. To be fair, this problem is not confined to the graph: most visualization techniques do not scale very well. In particular, there is a limit to the amount of detail that can be effectively shown and discerned on a computer display; techniques such as *pan and zoom* facilitate the exploration of detailed visualizations, but fail to integrate local detail and global context and often confuse the user. In response, Furnas suggested an analogy to the wide angle or 'fisheye' lens: the *generalized fisheye view* [4]. By balancing local detail and global context, fisheye views (FEVs) display information at several levels of abstraction simultaneously. In this paper we describe a novel technique for generating FEVs of hierarchically nested graphs – a generalization of a graph that allows nodes to be recursively contained within other nodes; whereas a graph depicts binary relations on a set of elements, a nested graph can, in addition, express one-to-many relations [6].

Furnas' generalized FEV technique relied upon an abstract notion of a view *element* with an associated *a priori importance* (API), and a *distance* measure between any pair of elements. One element was selected to be the fisheye *focal point* (FP). A *degree of interest* (DOI) function computed a measure of the user's interest in each element; the DOI increased with API and decreased with distance to the FP. A FEV was obtained by displaying only elements whose DOI exceeded a threshold. The exact formulations of the DOI and distance functions were expected to vary with each application.

Recent applications of generalized FEVs can be loosely grouped into two categories: *filtering FEVs* and *distorting FEVs*.

Filtering FEV approaches use thresholding to constrain the display of information to relevant or interesting elements, and have remained more faithful to Furnas' original conception. For example, this approach was used to filter the display of aircraft maintenance data [12], and to control the amount of output produced in response to boolean queries [5]. Furnas described similar filtering FEVs of C programs and appointment calendars [4]. One graph FEV approach [3] described in the next section also belongs in this category.

Distorting FEV approaches, on the other hand, pursue the analogy to the fisheye lens further, but in doing so, they begin to depart from Furnas. The basic idea is to geometrically distort visualizations to obtain visual fisheye effects – rather than filtering elements, distortion alters elements' positions, shapes, and sizes. Two graph FEV approaches [11, 18] described in the next section are of this type.

*TreeMaps* [8, 21] and *Cone Trees* [16] are two techniques for visualizing hierarchical structures that incorporate FEV concepts by using shape and size distortion and 3-D perspective respectively. Although not strictly a distorting FEV, the *Perspective Wall*, used to display linear information [9], is a related visualization technique.

## 2 FEVs of graphs

Several approaches for obtaining FEVs of graphs have been documented [3, 11, 18, 19].

Fairchild *et. al* described three filtering FEVs of the 3D networks of SemNet: clustering, three-dimensional point perspective, and sampling density [3]; all three strategies were defined in terms of a single FP.

Sarkar and Brown described a distorting FEV of graphs in which a drawing of a graph was subjected to a distortion that altered the positions and sizes of nodes [18]. Although their solution could be implemented efficiently using special purpose graphics hardware, it was limited to non-nested graphs with a single FP; a multiple variable FP technique was described subsequently [19].

Misue and Sugiyama described three distorting FEVs: fisheye, orthogonal fisheye, and biform [11]. These methods have been applied to drawings of compound digraphs [20] – a nested graph formalism. The first two method employed distortions similar to Sarkar and Brown's, while the third used uniform scaling; all three methods supported multiple fixed strength FPs.

The main problem with the above approaches is their reliance on a geometric interpretation of distance. Under this assumption, the resulting FEV depends completely on the original drawing of the graph: elements positioned near a FP receive a higher DOI rating. Such approaches are therefore only effective in cases where the abstract distance between nodes corresponds to their proximity in the drawing (*e.g.,* transportation networks where nodes are positioned using cartographic data). This basic flaw was also noted in previous work [3, 18].

In this paper, we overcome this limitation by proposing a novel layout-independent FEV approach.

## 3 Layout-independent FEVs

The crux of our approach is to generate FEVs by changing the size of each element in relation to its DOI before it is positioned by a layout algorithm, rather than by altering an existing layout. Since our technique does not filter or distort existing layouts, it does not properly belong to either of the filtering and distorting FEV categories.

Before presenting our generalized FEV formulation and the generic FP proximity algorithm, we give definitions of basic terms and give a sketch of some graph layout and related concepts. The section concludes with examples and discussion.

### 3.1 Preliminaries

**Graphs**: A nested graph is $G = \langle G(V), G(A), G(C)\rangle$, where, $G(V)$ is a set of vertices, $G(A) \subseteq G(V) \times G(V)$ is a

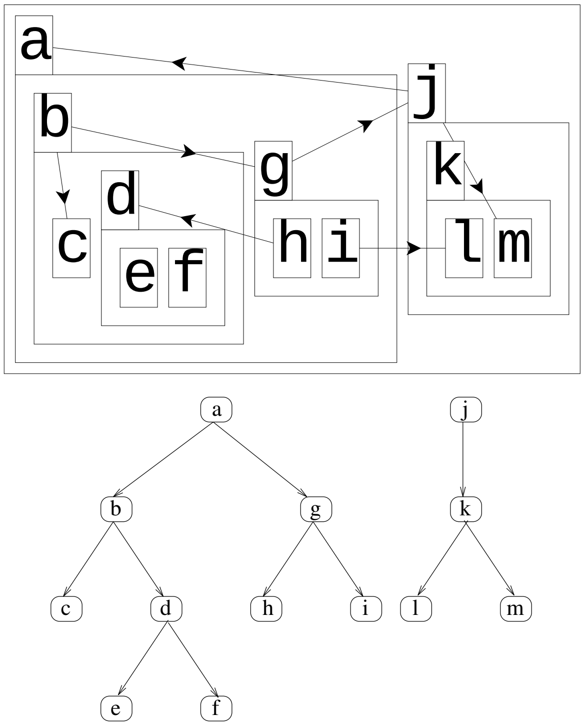


Figure 1: A sample graph and its nesting forest.

set of arcs, and $G(C) \subset G(V) \times G(V)$ is a set of containment arcs. Since we insist on hierarchical nesting, the nesting graph $G_n = \langle G_n(V) = G(V), G_n(A) = G(C), G_n(C) = \emptyset\rangle$ must be a forest of trees. A simple nested graph and its nesting forest are shown in Figure 1. For $v \in G_n(V)$, $anc(v)$ returns $v$'s ancestors, $desc(v)$ $v$'s descendants, $ch(v)$ $v$'s children, and $anc(v) \cap desc(v) = \{v\}$.

**Vertex geometry**: For layout purposes, the geometry of a vertex is specified relative to the geometry of its parent in $G_n$. This makes it possible to resize all vertices in the subtree rooted at $v$, simply by changing the extents of $v$. The bounding box of $v$ is computed as the smallest rectangle that fully encloses $v$'s label and subgraph of contained vertices.

**Graph layout**: In drawing nested graphs, topological nesting is naturally depicted by visual containment. A simple, bottom-up algorithm is:

```
1. foreach v ∈ G_n(V) in postorder wrt G_n(A)
2.     layout graph ⟨ch(v), (ch(v) × ch(v)) ∩ G(A), ∅⟩
3.     compute bounding box for v
4. endfor
5. layout non-nested vertices
```

Step 2 positions all vertices nested within a given vertex relative to one another. Similar layout algorithms have been described in the literature [3, 7, 10, 13, 20]. As an aside, notice that a composite layout strategy [7] can be realized simply by associating a specific layout algorithm with each vertex.

### 3.2 Generalized FEVs of nested graphs

In this formulation, the abstract view elements are vertices, thus $F \subseteq G(V)$ is a set of FPs. The DOI of $v \in G(V)$ is expressed as:

$$doi(v) = api(v) \cdot (p \cdot wt(v) + (1-p) \cdot proxim(v))$$

The functions $api$, $wt$ and $proxim$ return non-negative real values: $api(v)$ is $v$'s API, $wt(v)$ is a generic measure of $v$'s proximity to the FPs, $proxim(v)$ is an application specific measure of $v$'s proximity to the FPs, and $0 \leq p \leq 1$ reflects the tradeoff between generic and domain-specific notions of distance. Note that $api$ and $proxim$ are fixed by the semantics of the application, while $wt$ is computed by our algorithm. To generate a FEV, the following line must be inserted after Step 3 in the nested layout algorithm:

uniformly scale the bounding box of $v$ by $\sqrt{doi(v)}$

which increases the total area of $v$'s bounding box by $doi(v)$, consistent with the interpretation that the amount of screen space allocated to an element should reflect its DOI. The function $mag$ maps a FP to its non-negative real-valued magnification. For example, $mag(f) = 4.0$ implies that the area allocated to vertex $f \in F$ in the FEV should be four times greater than what it would have been in a regular view.

### 3.3 Generic FP proximity algorithm

Given $G_n, F, api$, and $mag$, the algorithm determines a mapping $wt$ such that

$$\forall\, f \in F, \prod_{v \in anc(f)} wt(v) = mag(f).$$

Note that due to the relative geometry of vertices, the final size of a vertex depends on the product of the weights of its ancestors. The above formula, therefore, says that in the absence of domain-specific distance metrics (*i.e.,* $p = 1$ in $doi(f)$), and in addition to $api(f)$ contribution, each FP $f$ will have its total area magnified by $mag(f)$.[1] The trick, then, is to distribute the value $mag(f)$ among the ancestors of $f$ in such a way that the resulting layout will exhibit the desired FEV characteristics. Now, we present the algorithm.

**Step 1**: First, for each vertex $v \in G(V)$ we compute $key(v) = desc(v) \cap F$. Figure 2 shows the keys associated with the vertices in the sample graph given $F = \{c, f, h, m\}$. Vertices that share the same key form a path in the nesting forest, and are associated with the same FP: the path's sink, $s \in G_n(V)$; if $s \notin F$, $s$ is a composite FP (a least common ancestor of two or more FPs). In our example, there are two composite FPs: $a$ and $b$. The mapping $fpSet(f)$ returns the set of vertices associated with FP $f$. For example, $fpSet(m) = \{j, k, m\}$ and $fpSet(f) = \{d, f\}$.

[1] Note that $0 < mag(f) < 1$ could also be used to explicitly reduce $f$'s area.

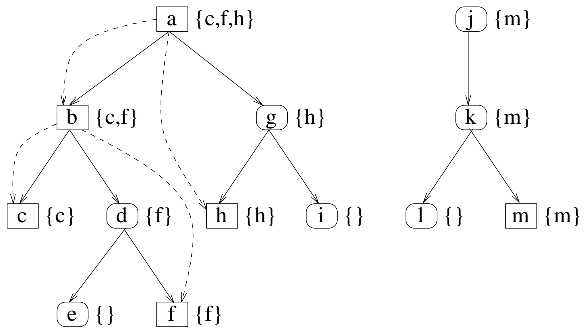


Figure 2: The marked-up nesting forest of the sample graph. The labels of FP vertices have square corners while the rest have rounded corners.

**Step 2**: Having obtained the keys, we generate the FP containment graph $G_c = \langle G_c(V), G_c(A), \emptyset\rangle$. Each $v \in G_c(V)$ is a FP that either contains one or more FPs or is contained by a FP. $G_c(A) \subset G_c(V) \times G_c(V)$ is the set of $(u, v)$ arcs such that $u$ contains $v$ and there is no $w$ for which $u$ contains $w$ and $w$ contains $v$. Following our example, $G_c$ is the subgraph induced by the dashed arcs in Figure 2.

**Step 3**: We now compute the values $mag(f)$ for each composite FP $f$ in a single postorder traversal of $G_c$. The magnification of each composite FP is a function of the set of FPs that it contains. For example, $mag(b) := \mu(\{\pi(c, b), \pi(f, b)\})$, where

$$\pi(u, v) = \prod_{w \in fpSet(u) \cup \{v\}} \delta(w, u, anc(u))$$

computes the product of vertex weights along the $v$ to $u$ path assuming that $F = \{u\}$, and $\delta(v, f, S)$ is a function that assigns a weight to $v$ assuming that $mag(f)$ is distributed among the vertices in $S$. For the examples shown in this paper, $\mu(S)$ returns the minimum element in $S$, while $\delta(v, f, S)$ distributes $mag(f)$ evenly among all of the vertices in $S$ (*i.e.,* $\delta(v, f, S) = mag(f)^{\frac{1}{|S|}}$, $\forall\, v \in S$).

Next, with a single postorder traversal of $G_c$, for every $(u, v) \in G_c(A)$, we scale FP magnifications as follows: $mag(v) := mag(v)\,/\,mag(u)$.

**Step 4**: Now we are ready to compute $wt$. First, $wt$ is initialized as follows: $\forall\, v \in G(V)$, $wt(v) := 1.0$. Then, using the scaled magnification values, for every FP $f$ (both

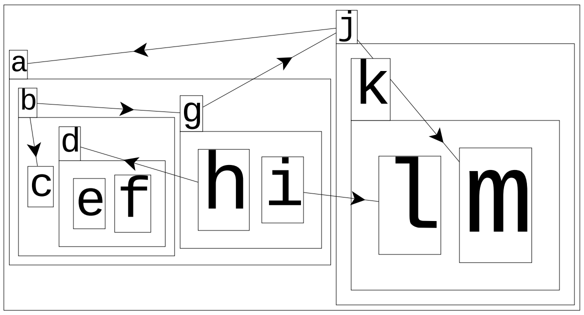


Figure 3: A FEV of the sample nested graph.

composite and original) and for every vertex $v \in fpSet(f)$, we compute $wt(v) := \delta(v, f, fpSet(f))$.

### 3.4 Examples

Figure 3 shows a FEV of the sample graph with FPs $\{c, f, h, m\}$, having magnifications $2, 4, 8$ and $16$, respectively. A large 'real-life' example, an object code memory overlay diagram and its FEV are shown in Figure 4. The overlay diagram depicts a nested series-parallel network that specifies how a large object code program is partitioned into independent overlay modules to improve run-time performance by minimizing disk thrashing [2]. Although a zoom feature could be used to study individual portions of this diagram, a FEV allows the user to explore several areas simultaneously without sacrificing context.

### 3.5 Discussion

Having described the algorithm, several points require further clarification.

While most FEV algorithms compute the distance component of DOI for every element in the structure, our algorithm distributes a FP's magnification among its ancestors. This decision can be justified by observing that since the final size of a vertex depends on the weights of its ancestors, spreading $mag(f)$ among vertices in $anc(f)$ results in the fortuitous side-effect of generating a FEV.

Although our weight assignment algorithm uses only the nesting information $G(C)$ and ignores the arcs $G(A)$, the arc information could be reflected in the DOI through the application specific proximity mapping $proxim$.

Superficially our approach bears some resemblance to TreeMaps[8, 21], in which a hierarchy is depicted by nested rectangles – the size of a rectangle reflects its DOI (specified as a fraction of its parent's DOI). There are, however, a number of differences. TreeMaps, a hierarchy viewing technique: (1) employs a fixed top-down layout algorithm which is tuned for efficient usage of screen space by arbitrarily changing the shapes of vertices, (our bottom-up approach supports composite general graph layout algorithms); (2) does not explicitly differentiate between the API and distance components of the DOI (we do); and most importantly, (3) does not attempt to compute the DOI of each vertex given a set of FPs. It would be fair to suggest that our algorithm could be adapted to automatically determine the DOI values for use with TreeMaps.

## 4 Implementation

Most of the figures shown in this paper were produced by a set of Unix programs [14] developed at the University of Toronto. The entire software package comprises approximately 12000 lines of C++ source code, and includes programs that can perform composite graph layout of nested graphs, produce several types of FEV visualizations, and convert graphs into encapsulated PostScript files. This software has been incorporated into several systems, including Hy+ [1] and 4Thought [2, 17].

## 5 Future directions

The approach outlined in this paper suggests several directions for future research.

Although the simple formulations of the vertex weight distribution function $\delta$ and the composite FP magnification function $\mu$ seem adequate in the absence of domain-specific distance metrics, alternative formulations are currently being studied. For example, rather than distributing weights evenly, a formulation of $\delta(v, f, S)$ that increases the weights of vertices near $f$ while decreasing the weights of others may produce better FEVs. Similarly, alternative formulations of $\mu$ may yield more aesthetically pleasing results.

A FEV strategy should not be regarded simply as a way to generate static visualizations, but as an integral part of a highly interactive process. This application suggests the need for an incremental version of the algorithm. In the simplest case, such an algorithm should provide efficient means to handle insertions and deletions of FPs, by minimizing the number of times that layout must be repeated. Furthermore, special cases such as the deletion followed by insertion of same FP (*i.e.,* FP migration), should be investigated.

In an interactive application, the problem of layout stability [13] may arise in the form of FP updates that cause unpredictable changes in the resulting layout, as many layout algorithms are sensitive to the sizes and shapes of vertices (*e.g.,* a grid layout that computes the size and shape of each grid cell based on the dimensions of vertices). A simple approach would be to force subsequent re-layout to use original layout settings at the risk of producing less effective layouts; a more challenging solution would be to

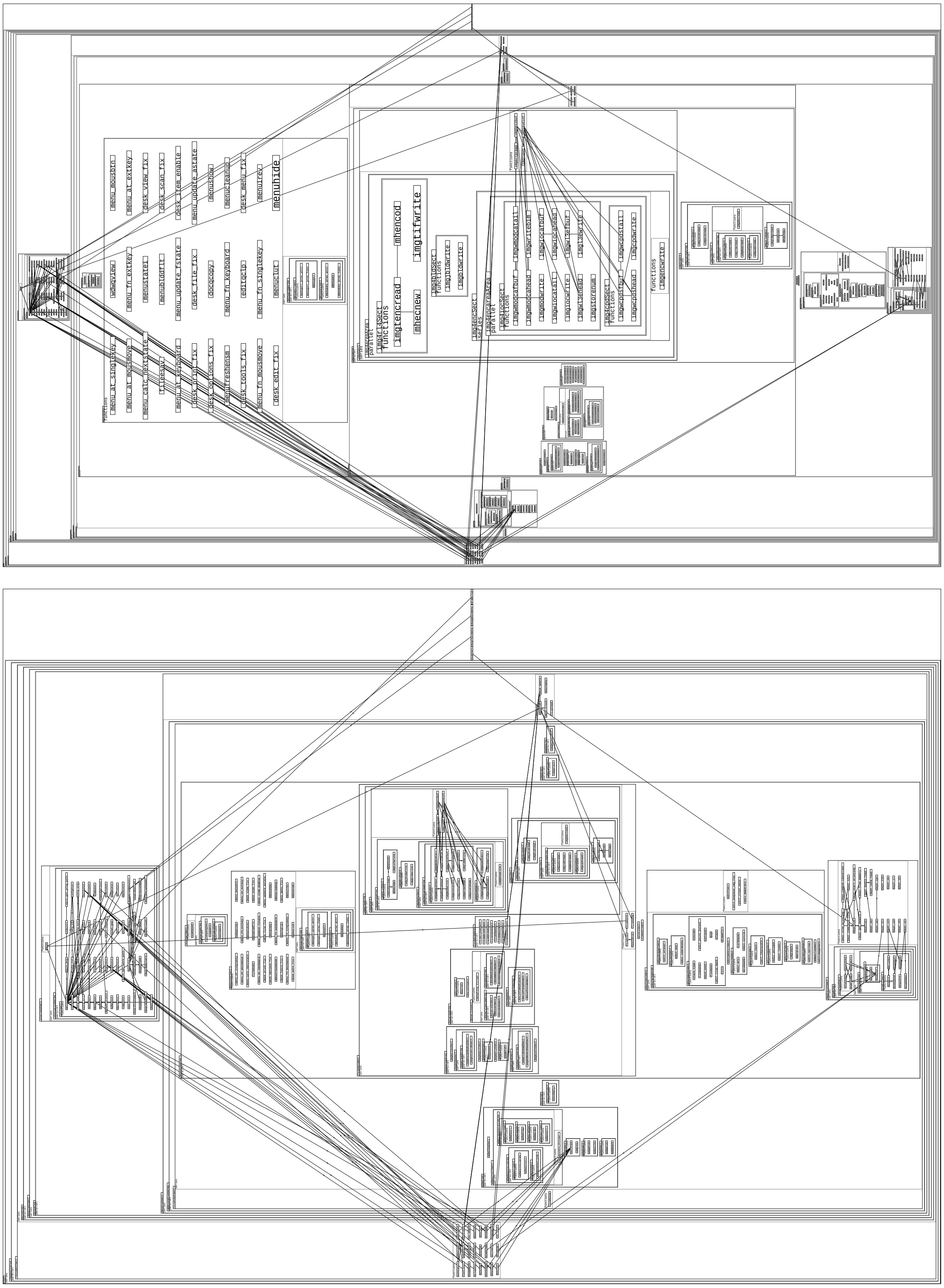

Figure 4: A code overlay diagram (bottom), and its FEV given $F = \{menuhide, imgtifwrite\}$.

determine modified layout settings that result in a measurable compromise between stability and effectiveness.

## 6 Conclusion

Although the graph can be a helpful tool in the study of complex relationships, drawings of large graphs can be difficult to comprehend. As one strategy for coping with this problem, we described a generalized approach for generating layout-independent FEVs of nested graphs with multiple variable focal points, and presented an algorithm that computes FEVs in the absence of application specific distance metrics. Our approach can be applied to most problems involving hierarchically nested structures and does not suffer the shortcomings of existing solutions. In particular, since our technique does not rely on geometric notions of distance, it is not sensitive to the original layout of a graph and thus its use is not limited to graphs whose layouts are based on cartographic criteria.

### Acknowledgements

This work is a part of on-going research in relational data visualization within the database group at the University of Toronto. I am grateful to Alberto Mendelzon and other members of this group for their contributions. I would also like to thank the anonymous reviewers for their many constructive criticisms and suggestions.

## References

[1] M.P. Consens, I.F. Cruz, and A.O. Mendelzon. Visualizing Queries and Querying Visualizations. *ACM SIGMOD Record*, 21(1):39–46, Mar. 1992.

[2] M.P. Consens, A.O. Mendelzon, and A.G Ryman. Visualizing and Querying Software Structures. In *14th Intl. Conf. on Software Engineering*, pages 138–156, 1992.

[3] K.M. Fairchild, S.E. Poltrock, and G.W. Furnas. SemNet: Three-Dimensional Graphic Representations of Large Knowledge Bases. In R. Guindon, editor, *Cognitive Science and its Applications for Human-Computer Interaction*, pages 201–233. Lawrence Erlbaum Associates, 1988.

[4] G.W. Furnas. Generalized Fisheye Views. In *ACM CHI '86*, pages 16–23, Boston, MA, Apr. 1986. ACM.

[5] R. Godin, J. Gecsei, and C. Pichet. Design of a browsing interface for information retrieval. In *ACM SIGIR '89*, pages 32–39. ACM, Jun. 1989.

[6] D. Harel. On Visual Formalisms. *Communications of the ACM*, 31(5):514–530, May. 1988.

[7] T.R. Henry and S.E. Hudson. Interactive Graph Layout. In *UIST '91*, pages 55–64. ACM, 1991.

[8] B. Johnson and B. Shneiderman. Tree-Maps: A Space-Filling Approach to the Visualization of Hierarchical Information Structures. In *IEEE Visualization '91*, pages 284–291, 1991.

[9] J.D. Mackinlay, G.G. Roberson, and S.K. Card. The Perspective Wall: Detail and Context Smoothly Integrated. In *ACM CHI '91*, pages 173–179. ACM, Apr. 1991.

[10] E.B. Messinger, L.A. Rowe, and R.R. Henry. A Divide-and-Conquer Algorithm for the Automatic Layout of Large Directed Graphs. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(1):1–12, 1991.

[11] K. Misue and K. Sugiyama. Multi-viewpoint Perspective Display Methods: Formulation and Application to Compound Graphs. In *4th Intl. Conf. on Human-Computer Interaction*, volume 1, pages 834–838, Stuttgart, Germany, Sep. 1991. Elsevier.

[12] D.A. Mitta. A Fisheye Presentation Strategy: Aircraft Maintenance Data. In *INTERACT '90*, pages 875–880. IFIP, Elsevier, 1990.

[13] F. Newbery Paulisch and W.F. Tichy. EDGE: An Extensible Graph Editor. *Software–Practice and Experience*, 20(S1):63–88, Jun. 1990.

[14] E.G. Noik. Unix Filters to Draw and Manipulate GXF Graphs. In A.O. Mendelzon, editor, *Declarative database visualization: recent papers from the Hy+/GraphLog project*. Forthcoming technical report, CSRI, U. of Toronto, 1993.

[15] J. Ramanathan and R.L. Hartung. A Generic Iconic Tool for Viewing Databases. *IEEE Software*, 6(9):50–57, Sep. 1989.

[16] G.G. Robertson, J.D. Mackinlay, and S.K. Card. Cone Trees: Animated 3D Visualizations of Hierarchical Information. In *ACM CHI '91*, pages 189–194. ACM, Apr. 1991.

[17] A.G. Ryman. Foundations of 4Thought. In *CAS '92*, pages 133–155, Toronto, Canada, Nov. 1992.

[18] M. Sarkar and M.H. Brown. Graphical Fisheye Views of Graphs. In *ACM CHI '92*, pages 83–91, Monterey, CA, May. 1992. ACM.

[19] M. Sarkar and S.P. Reiss. Manipulating Screen Space with *Stretch Tools:* Visualizing Large Structure on Small Screen. Technical Report CS-92-42, Dept. of Comp. Sci., Brown U., Sep. 1992.

[20] K. Sugiyama and K. Misue. Visualization of Structural Information: Automatic Drawing of Compound Digraphs. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(4):876–892, 1991.

[21] D. Turo and B. Johnson. Improving the Visualization of Hierarchies with Treemaps: Design Issues and Experimentation. In *IEEE Visualization '92*, pages 124–131, Boston, MA, 1992.